Homework 2: Q3

Name: Waiwai Kim, waiwaiki

1 Part (a) Proof Idea

We will use the examples and algorithms provided in the week 4 recitation note in order to provide counter example of greedy algorithms that do not work properly to truncate the schedule.

Table 1: Example Schedule

	t_1	t_2	t_3	t_4
S_1	E_1	0	E_2	0
S_2	0	E_1	0	E_2

First, we will apply the engineer-centric greedy algorithm in Figure 1 to the schedule shown in Table 1.

- Per E_1, we notice that E_1 meet with S_2 at t_2 as their last meeting.
- Per E_2, we notice that E_2 meets with S_2 as their last meeting. The algorithm truncates here.

Truncating the both meetings is not possible. Because truncating at (S_2, E_1) at t_2 means that S_2 is not available at t_4. Thus, truncating based on the second bullet point is not possible.

Figure 1: Greedy Algorithm - Engineer Centric

```
        //input: n is the number of students and engineers
        //input: m is the number of time slots
        //input: student schedules are S[i] where 0 < i ≤ n and is of length m
        //input: engineer schedules are E[i] where 0 < i ≤ n and is of length m
 5.     //input: each index of the arrays will be 0 if the slot is free, or will
        //       contain a number corresponding to a scheduled engineer or student
        //return will be a list of pairs of engineers and students

        ret = []
10.
        for i from 1 to n:
            for j from m-1 to 0:
                if E[j] != 0:
                    ret.append((E[j],i))
15.                 break

        return ret
```

Similarly, we will apply the student-centric greedy algorithm in Figure 2 to the schedule shown in Table 1. Note that both students have E_2 as their last meeting of the day. This means E_2 is matched with two students. This is not possible. Thus, the student-centric greedy algorithm does not work.

- Per S_1, we notice that S_1 meet with E_2 at t_3 as their last meeting.

- Per S_2, we notice that S_2 meets with E_2 at t_4 The algorithm truncates here.

Figure 2: Greedy Algorithm - Student Centric

```
        //input: n is the number of students and engineers
        //input: m is the number of time slots
        //input: student schedules are S[i] where 0 < i ≤ n and is of length m
        //input: engineer schedules are E[i] where 0 < i ≤ n and is of length m
 5.     //input: each index of the arrays will be 0 if the slot is free, or will
        //       contain a number corresponding to a scheduled engineer or student
        //return will be a list of pairs of engineers and students

        ret = []
10.
        for i from 1 to n:
            for j from m-1 to 0:
                if S[j] != 0:
                    ret.append((i,S[j]))
15.                 break

        return ret
```

2 Part (b) Algorithm Idea

Per each schedule, either students' or engineers', the algorithm has to determines the last interview meeting. After the lasting meeting for student A and Engineer B at time t, student A and Engineer B will not be available. The last meeting is "stable" if it results in NO student getting stood-up.

We can reduce this problem to a stable matching problem that we have seen in classes.

- student = male (or vice versa)
- engineer = female (or vice versa)
- instability = getting stood-up
- the resulting set of matches = perfect matches in which every alternative match does not have bilateral incentives to break up their current marriages = perfect matches of students and engineers in which no student gets stood-up

Note that the notion of instability or getting stood-up means that some student$_i$ tries to meet with (aka proposes to) an engineer$_j$ that has already left the building after meeting with student$_k$. This is an instance of instability because engineer$_j$ has a meeting scheduled with student$_i$ after meeting student$_k$ (aka instbility exists because engineer$_j$ "prefers" student$_i$ to student$_k$, while student$_i$ "prefers" engineer$_j$ to some engineer he has just met previously).

Based on the aforementioned reduction, we will construct two preference lists or calendar meeting notices.

- student ranks engineer in chronological order
- engineer ranks student in reverse chronological order

Once we have the preference lists, we can run the Gale-Shapley algorithm in which students proposes engineer to meet.

3 Part (b) Algorithm Details

let's run the algorithm on the example provided in Table 2 and 3.

- S_1 matched with E_1 because E_1 is free.
- S_2 matched with E_1, and (S_1, E_1) broken because E_1 prefers S_2 to S_1.
- S_3 matched with E_2 because E_2 is free.
- S_1 matched with E_2, and (S_3, E_2) is broken because E_2 prefers S_1 to S_3.
- S_3 matched with E_3 because E_3 is free.
- the resulting set of last meeting is = [(S_1,E_2), (S_2, E_1), (S_3,E_3)]

The resulting set is indeed stable, and no one gets stood-up. Working with an example of a small size allows us to double-check the result thoroughly and manually.

Table 2: Students Schedule Preference List - Chronological

	t_1	t_2	t_3	t_4
S_1	E_1	0	E_2	E_3
S_2	0	E_1	E_3	E_2
S_3	E_2	E_3	E_1	0

Table 3: Engineer Schedule Preference List - Reverse Chronological

	t_4	t_3	t_2	t_1
E_1	0	S_3	S_2	S_1
E_2	S_2	S_1	0	S_3
E_3	0	S_1	S_2	S_3

If you express the steps shown above in terms of a set of inputs and pseudo code, you will have the following 1.

- input: n is the number of students and engineers.
- input: m is number of time slots.
- input: student schedules are S[i] where 0<i≤n and is of length m. For a given student$_i$, S[i] ranks n engineers in chronological order.
- input: engineer schedule are E[i] where 0<i≤n and is of length m. For a given engineer$_i$, E[i] ranks n students in reverse chronological order.
- output: a list of pairs of engineers and students.

Algorithm 1 Algorithm1

```
1:  procedure ALGORIGHM-1(Inputs)
2:      return_list = []
3:      while ∃ a free student who has been matched: do
4:          let engineer E be the earliest the student can meet that he or she has not sent a meeting notice yet
5:          for i from 1 to n: do
6:              student sends a meeting notice to engineer
7:              if engineer is free: then
8:                  engineer accepts the meeting notice
9:              else engineer is already matched wit another student:
10:                 if if student meeting time is chronologically later than the current meeting time: then
11:                     cancel the meeting with the currently matched student.
12:                     set up a new meeting with the student
13:                 else student meeting time is chronologically before the current meeting time
14:                     Pass
15:                 end if
16:             end if
17:         end for
18:     end while
        return return_list
19: end procedure
```

4 Part (b) Proof Idea

We will prove by contradiction that the set of matching output by the algorithm provided above is stable. If a matching is not valid, this means that a student does not have a distinct engineer as his or her last meeting. In other words, for some student S_i is matched with an Engineer E_j after student S_k has already met with E_j and left the building. However, S_i "prefers" E_j to its current engineer, and E_j "prefers" S_i to S_k. This contradicts the assumption for a stable matching between student and engineer.

5 Part (b) Proof Details